AN3007

megaAVR® 0 系列上的 FreeRTOS 入门指南

特性

- FreeRTOS™的配置和基本特性
- 调试和典型错误
- Atmel | START 示例代码

简介

作者：Eira Mørch-Thoresen，Microchip Technology Inc.

FreeRTOS 是嵌入式器件的实时操作系统内核。它采用小巧的设计，简单易用，仅包含几个主要用 C 语言编写的文件。

单片机通常用于实时嵌入式应用，这意味着嵌入式系统必须能够在严格定义的时间内响应某些事件。为了确保系统满足这些时间节点的要求，RTOS 提供了一个调度程序，用于决定在某个时间实例运行哪个任务。

FreeRTOS 针对任务、任务通信和调度提供了多种功能，并且已成为单片机实时操作系统（Real-Time Operating System，RTOS）的公认标准。FreeRTOS 的主要设计目标是稳健性、易用性和小尺寸。

本文档首先介绍如何配置 FreeRTOS，然后介绍阻断功能、任务间通信方案和调度。在介绍调试相关的信息后，还会提供演示代码。此外，本应用笔记还为演示中的每个任务提供了 UML 图。

目录

1. 相关器件

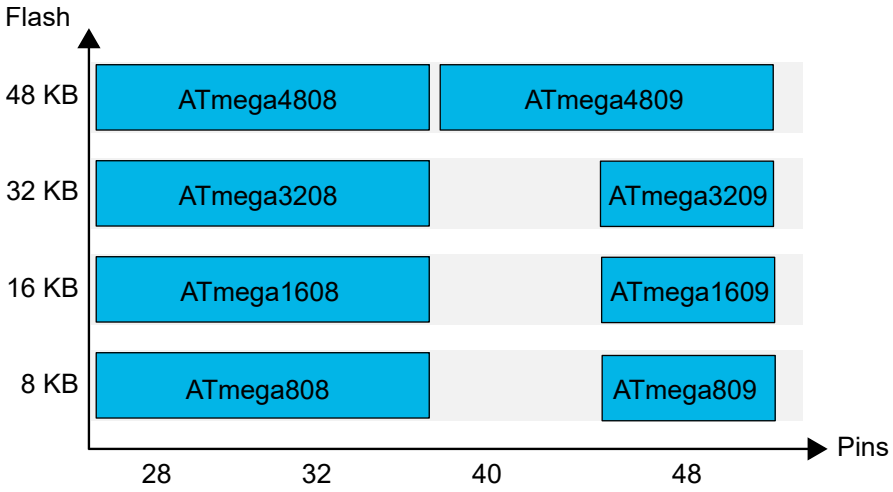
本章列出了文中涉及的相关器件。

1.1 megaAVR 0 系列

下图所示为 megaAVR 0 系列器件，注明了不同的引脚数与存储器大小：

- 垂直迁移无需修改代码，因为这些器件的引脚和功能完全兼容。
- 从右到左迁移会减少引脚数，进而减少可用的功能。

图 1-1. megaAVR® 0 系列概览



具有不同闪存大小的器件通常也具有不同的 SRAM 和 EEPROM。

2. 从 Atmel | START 开始

创建 FreeRTOS 应用程序最简单的方法是，从 Atmel | START 下载示例，然后在此基础上进行修改。这样可以保证不会缺少任何 FreeRTOS 文件，且所有文件都可正常使用。然后可以删除演示代码，并开始应用程序开发。

本应用笔记将使用以下 Atmel | START 示例项目：ATmega4809 FreeRTOS 示例。

3. 配置 FreeRTOS

FreeRTOS 使用名为“FreeRTOSConfig.h”的配置文件。通过更改此文件，可以自定义 FreeRTOS 应用程序，使其按预期的方式运行。配置文件必须包含在预处理器路径中。

通常需要更改的配置是存储器大小、堆栈选项以及时钟和节拍率。此外还可以配置任务优先级和抢占。配置文件包含很多选项，本应用笔记仅涵盖一部分可用选项。要查看所有选项，请参见 https://www.freertos.org/a00110.html。

3.1 配置时钟和节拍率

实时操作系统（RTOS）使用系统节拍，该节拍是定时器的时间单位。FreeRTOS 使用单片机的 TCB0 定时器来产生自己的节拍中断。FreeRTOS 内核使用节拍测量时间，每次出现节拍时，调度程序都会检查任务是否应被唤醒或取消阻断。

必须配置 `configCPU_CLOCK_HZ` 定义，以确保 FreeRTOS 时序正确。应在此输入 TCB0 时钟的频率。默认情况下，TCB0 的运行频率和“相关器件”章节中所列器件的 CPU 频率相同。

`configTICK_RATE_HZ` 用于确定 RTOS 节拍中断的频率。由于每次出现节拍时，都会产生一些 CPU 开销，因此与 CPU 频率相比，节拍频率不应设置得过高。

如果有多个任务具有相同的优先级，则当 `configUSE_TIME_SLICING` 置 1 时，RTOS 将于每次出现节拍时在这些任务之间切换。因此，使用的节拍率越高，每个任务的 CPU 时间就越短。例如，如果使用的节拍率为 1000 Hz，则每个任务的时间片将为 T=1/f=1/1000 Hz，也就是 1 ms。

3.2 配置存储器

FreeRTOS 具有两种存储器分配方案：静态分配和动态分配。静态存储器分配要求应用程序自行分配存储器和取消分配存储器。这种分配可能更难以实现，但可以提供更多控制。当使用动态分配方案时，存储器分配会在 API 函数中自动进行。应用程序仅需调用创建和删除函数。要使用动态分配，在配置文件中将 `configSUPPORT_DYNAMIC_ALLOCATION` 定义置 1。

使用动态分配时，FreeRTOS 需要为称作堆的区域分配一部分 RAM。以“Create”结尾的 FreeRTOS 函数在堆上分配存储器。在调用 `xTaskCreate()` 创建由栈和任务控制块（Task Control Block，TCB）组成的任务后，将针对该任务在堆上分配存储器。任务栈的大小在创建任务时定义。创建时，队列、互斥和信号量也将分配堆存储器。有关定义正确栈大小的提示，请参见“在 FreeRTOS 中调试”部分。参见下图以了解 FreeRTOS 存储器的工作方式。

使用动态分配时，`configTOTAL_HEAP_SIZE` 需要足够大。换句话说，它必须足够大才能适应所有任务栈和其他已分配的存储器。在演示示例中，堆大小设置为 0x800（2048 字节）。为了优化堆大小，可以使用 `xPortGetFreeHeapSize()`，此函数会返回未分配堆的大小。有关如何找到正确堆大小的提示，请参见“在 FreeRTOS 中调试”部分。

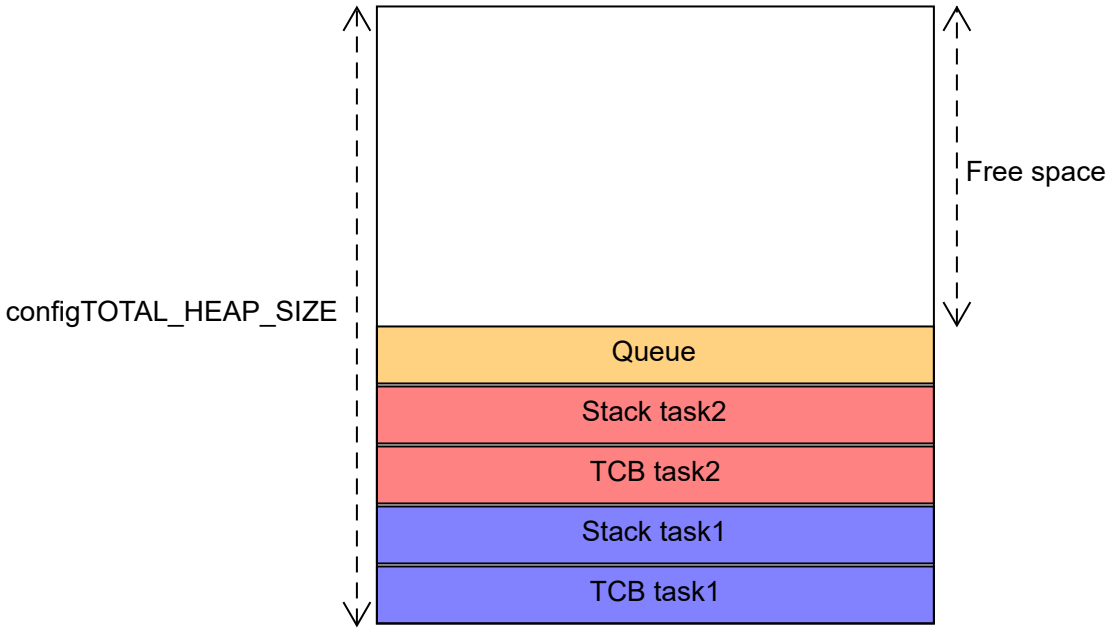
FreeRTOS 中提供了五种不同的堆实现，称为堆 1-5。要选择使用哪种堆分配方式，可在 `heap.c` 文件中更改 `USE_HEAP` 定义，该文件位于 `freeRTOS/portable/MemMang` 文件夹中。

Atmel | START 示例使用堆 1 实现。堆 1 与静态分配方式类似，一旦存储器被占用，就无法释放或重新分配。堆 1 易于调试，但需要在应用程序的整个生命周期内，将任务和其他 FreeRTOS 对象（例如队列、信号量和互斥）保留在堆上，以避免创建和销毁这些对象而导致应用程序耗尽存储器。

与堆 1 不同，堆 2 方案允许释放存储器，但如果分配的存储器大小是随机的，则应小心使用，因为这可能导致可用空闲存储器碎片化，从而引起分配故障。

堆 3、堆 4 和堆 5 是更高级的方案。更多信息，请参见 www.freertos.org/a00111.html。

图 3-1. FreeRTOS 存储器



3.2.1 配置栈

在 FreeRTOS 中，每个任务都有自己单独的栈。使用动态存储器分配时，会在堆上分配任务栈。除空闲任务外，也会在调用 `xTaskCreate()`时分配栈，其中的一个输入参数就是任务的栈大小。

调用 `vTaskStartScheduler()`时会创建空闲任务，以确保始终至少有一个任务能够运行。空闲任务的栈大小由配置文件中的 `configMINIMAL_STACK_SIZE` 给定。

重要的一点是，已分配的栈存储器与从队列、信号量等分配的存储器之和不超过 `configTOTAL_HEAP_SIZE` 中定义的堆大小。

4. 从 RTOS 开发人员角度思考

实时编程及其严格的时间限制改变了开发人员的思维方式。将应用程序划分为多个任务，而不是使用典型的状态机实现。一个任务负责应用程序的一部分，而另一个任务负责其他部分。例如，在 Atmel | START 示例中，一个任务负责使 LED 闪烁，另一个任务负责将时间写入 OLED 屏幕，其他几个任务分别负责不同的部分。在实时系统中，这些任务同时运行，每当调度程序允许时，即会执行其中一部分代码。这与典型的线性应用程序代码不同，在线性应用程序代码中，所有内容都按顺序且始终按相同顺序进行评估。

如果有些任务比其他任务更重要并且需要更快的响应时间，则可以将 RTOS 配置为允许较高优先级的任务中断较低优先级的任务。这称为抢占。

为了使应用程序正确运行，重要的是要确保各任务不会在可能造成损害的时候彼此中断。例如，在某个任务完成对共享资源的计算之前，其他任务先读取了该共享资源的值。然后，读取值可能是错误的，这又可能导致其他问题。重要的一点是，要清楚如何使用任务通信来确保互斥，以避免出现此类错误。有关信息，请参见任务通信部分。

下图显示了一个简单的自制程序的执行顺序，以及从线性编程到并行或并发任务的转换过程。

图 4-1. 线性编程

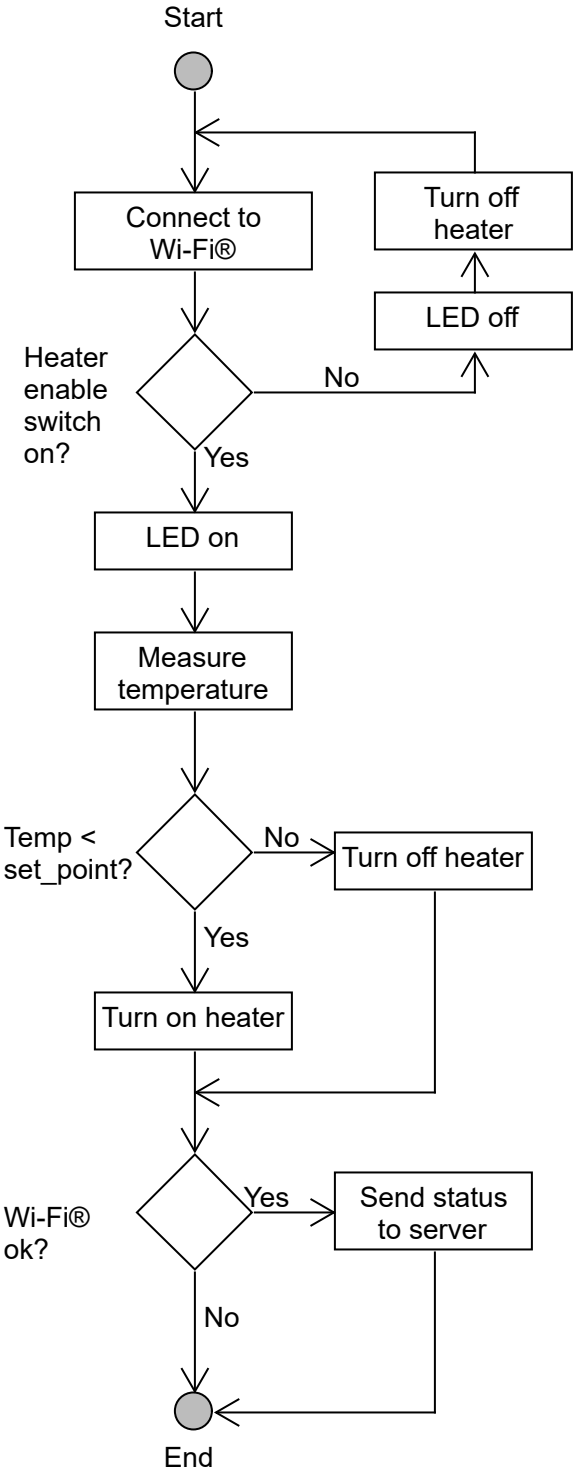
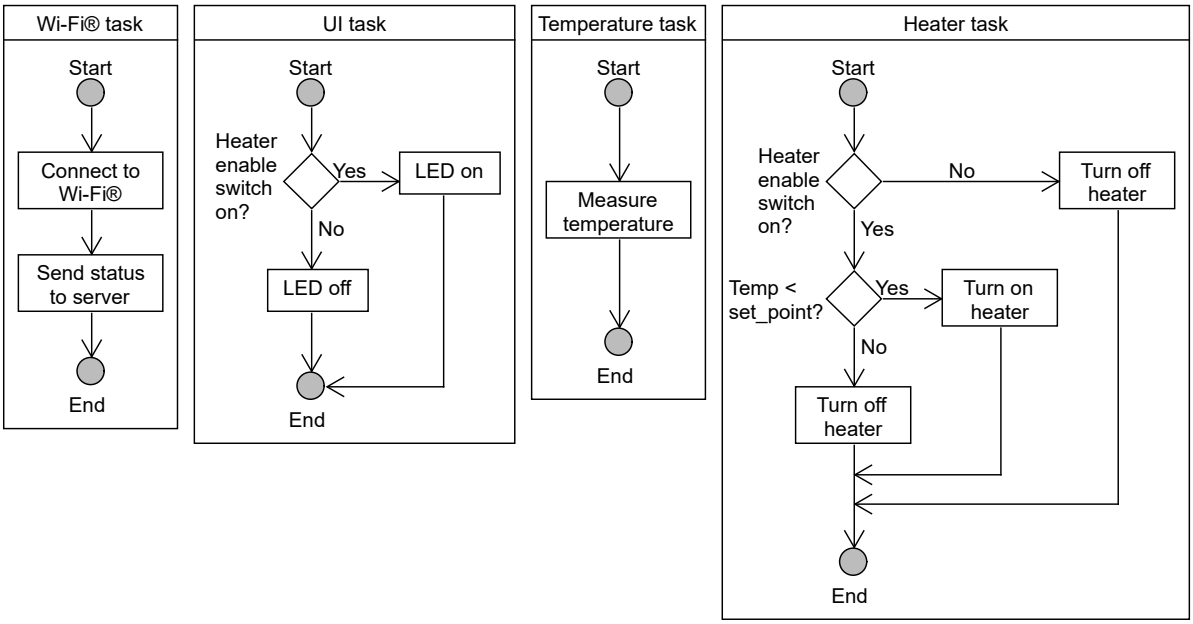

Start
Connect to Wi-Fi®
Turn off heater
LED off
Heater enable switch on?
No
Yes
LED on
Measure temperature
Temp < set_point?
No
Turn off heater
Yes
Turn on heater
Wi-Fi® ok?
Yes
Send status to server
No
End

图 4-2. 任务驱动代码



4.1 任务

通过调用函数 `xTaskCreate(taskName, pcName, stackDepth, pvParameters, priority, pxCreatedTask)` 创建任务。任务通常实现为无限循环。请参见以下示例来了解如何实现任务。

```
xTaskCreate(exampleTask, "example", configMINIMAL_STACK_SIZE, NULL, 1, NULL);

void exampleTask(void *pvParams){
    while(1){
        //do task work
    }
}
```

4.2 阻断与非阻断功能

阻断功能可阻止任务继续执行。当某个任务被阻断后，RTOS 会转向执行其他任务。在这种情况下，永远不会存在任何循环，且 CPU 不执行任何操作，因此这会优化 CPU 利用率。

通常，当任务调用延时函数、等待通信或其他任务使用的资源或者调用中断（ISR）时，任务将被阻断。调用 `vTaskDelay(ticksToDelay)` 时，任务将被阻断指定数量的节拍。任务阻断的实际时间取决于在配置文件中指定的节拍率。

当多个任务共享资源时，它们必须等待其他任务完成对资源的使用，然后自己才能开始使用该资源。这种等待资源的情况也称作阻断。为了创建这种行为，可以使用信号量和互斥。下一节将更详细地介绍共享资源和同步。

4.3 任务通信

FreeRTOS 提供了五种主要的任务间通信机制：队列、信号量、互斥、流缓冲区和报文缓冲区。所有这些机制的共同点在于，它们可以在资源或数据不可用时将任务阻断。例如，如果某个任务想要将一个元素插入到已满队列中，它将最终等待（阻断）一段时间，然后再次检查队列中是否有任何可用空间。

4.3.1 队列

队列提供用户可自定义的固定长度的任务间通信。开发人员可在创建队列时指定报文长度。通过调用 `QueueHandle_t queueName = xQueueCreate(queueLength, elementSize)` 即可完成此操作。输入参数 `queueLength` 指定队列中可以容纳的元素数量。`elementSize` 指定每个元素的大小（以字节数表示）。队列中所有元素的大小都必须相同。创建队列后，任务可通过队列发送和接收数据，从而实现相互通信。队列采用 FIFO 结构（先入先出），因此接收器将始终接收第一个插入的项。

用于发送到队列和从队列接收的函数如下：

```
xQueueSend(queueName, *itemToQueue, ticksToWait) xQueueReceive(queueName, *buffer,
ticksToWait)
```

最后一个参数 `ticksToWait` 指定在等待已满队列腾出空间时，发送任务应被阻断的时间。同样，它也指定在等待从空队列接收任何元素时，接收任务将被阻断的时间。此处可以使用定义 `portMAX_DELAY`。如果配置文件中的 `INCLUDE_vTaskSuspend` 置 1，则 `portMAX_DELAY` 将导致任务无限期地阻断（无超时）。如果置 0，则阻断时间将为 0xFFFF。

如果要从 ISR 调用发送和接收函数，则必须使用 `xQueueSendFromISR()` 和 `xQueueReceiveFromISR()`。

4.3.2 信号量

信号量用于同步和控制对任务之间共享资源的访问。信号量可以是二进制值，也可以是计数值，它本质上只是一个非负整数计数值。

二进制信号量初始化为 1，可用于保护一次只能由一个任务处理的资源。当某个任务使用资源时，信号量将递减为 0。如果其他任务想要使用该资源，并且信号量显示为 0，则该任务将被阻断。当使用资源完成第一个任务时，信号量将递增，从而可用于其他任务。可以通过 `SemaphoreHandle_t semaphoreName = xSemaphoreCreateBinary(void)` 创建二进制信号量。

计数信号量的工作方式与二进制信号量相同，但资源可由多个任务同时使用。例如，如果有一个停车场可以停放 10 辆车，则允许 10 次信号量访问。每次有车辆进入时，信号量都会递减 1，直至达到 0，并且在有车辆离开前不允许任何车辆进入。计数信号量应初始化为可同时访问资源的任务数量，可通过 `SemaphoreHandle_t semaphoreName = xSemaphoreCreateCounting(maxCount, initialCount)` 进行创建。

当某个任务需要受信号量保护的资源时，它会调用函数 `xSemaphoreTake(semaphoreName, ticksToWait)`。如果信号量评估为 0，则任务将被阻断 `ticksToWait` 中指定的时间。当使用信号量完成任务时，会调用函数 `xSemaphoreGive(semaphoreName)`。

4.3.3 互斥

互斥与二进制信号量极为类似，但不同之处在于，互斥提供了优先级继承机制。当高优先级任务被阻止访问低优先级任务占用的资源时，低优先级任务将继承高优先级任务的优先级，直到释放互斥为止。这样可以确保最大限度地缩短高优先级任务的阻断时间，因为其他中优先级任务此时无法抢占低优先级任务。

可通过使用 `semaphoreHandle_t mutexName = xSemaphoreCreateMutex(void)` 函数创建互斥。

由于优先级继承机制只适用于任务而不适用于中断，因此不应通过中断使用互斥。

4.3.4 流缓冲区

流缓冲区在两个任务之间或从 ISR 向任务传输数据。与队列不同，流缓冲区假定只有一个读取器和一个写入器。创建流缓冲区时，将指定缓冲区的最大大小。顾名思义，流缓冲区可用于在写入器和读取器之间传输字节流。字节流的长度可以是任意的，只要它在缓冲区的大小之内即可。

可使用函数 `xStreamBufferCreate(BufferSizeBytes, TriggerLevelBytes)` 创建流缓冲区。第一个输入参数指定缓冲区能够容纳的字节总数。第二个参数是触发级别，它指定在阻断的接收器退出阻断状态之前，缓冲区中必须保留的字节数。

如果缓冲区已满，发送器可能会被阻断。发送器任务在等待数据时应阻断的时间由发送函数中设置的超时给出。同样，如果缓冲区为空，接收器将被阻断。

与队列不同，在将报文放入缓冲区之前，发送器不需要准备好完整的报文。

发送和接收函数如下所示。

```
xStreamBufferSend(streamBuffer, *pvTxData, dataLengthBytes, ticksToWait);
xStreamBufferReceive(streamBuffer, *pvRxData, bufferLengthBytes, ticksToWait);
```

4.3.5 报文缓冲区

报文缓冲区的工作原理与流缓冲区极为相似，但不是让接收器请求一定数量的数据，而是由报文的第一部分指定报文长度。此外，只能读取指定长度的报文，而不能读取单个字节。

报文缓冲区是使用流缓冲区实现的，因此这里也适用只有一个读取器和一个写入器的情况。报文缓冲区的工作方式与流缓冲区也相同，因此，发送器将在缓冲区已满时阻断，而接收器在缓冲区为空时阻断。

可以使用函数 `xMessageBufferCreate(bufferSizeBytes)` 创建报文缓冲区。指定的大小应等于缓冲区能够包含的字节（而不是报文）总数。

使用以下函数可以完成数据的发送和接收。

```
xMessageBufferSend(messageBuffer, *pvTxData, xDataLengthBytes, ticksToWait);
xMessageBufferReceive(messageBuffer, *pvRxData, bufferLengthBytes, ticksToWait);
```

4.4 调度

调度是决定何时运行哪个任务的软件。FreeRTOS 在处理任务优先级时有两种工作模式：抢占模式和非抢占模式。可以在配置文件中设置使用哪种模式。当使用非抢占模式（也称作合作模式）时，由开发人员通过使用阻断函数和 `taskYIELD()` 函数来创建让出 CPU 的任务。

当使用抢占式调度程序时，如果优先级较高的任务变为非阻断状态，则另一任务将自动让出 CPU。但有一个例外情况：当从 ISR 阻断优先级较高的任务时，必须在 ISR 的末尾调用 `taskYIELD_FROM_ISR()` 函数才能进行任务切换。

如果 `configUSE_TIME_SLICING` 设置为 1，则调度程序也将在每次出现节拍时抢占同等优先级的任务。时间分片在合作模式下不可用。

在两种模式下，调度程序都将始终切换到优先级最高的非阻断任务。如果有多个非阻断的任务具有相同优先级，则调度程序将选择依次执行每个任务。此过程通常称为循环调度。

在抢占模式下，当优先级更高的任务进入非阻断状态时，会立即切换到这些任务。在合作模式下，释放信号量可能会使更高优先级的任务进入非阻断状态，但实际任务切换将只在当前执行任务调用 `taskYIELD()` 函数或进入阻断状态时发生。

有关调度的更多信息，请参见 Mastering the FreeRTOS Real Time Kernel - a Hands On Tutorial Guide 中的“调度算法”部分。

5. 在 FreeRTOS 中调试

本部分将介绍一些最常见的错误，并提供有关如何解决这些错误的信息。

5.1 堆调试

与堆有关的问题可能是由于分配的堆存储器大小太小，或者所选的堆实现不适合应用程序而引起的。

调用 `xPortGetFreeHeapSize()` 函数可以返回未分配的堆空间的总大小。如果该值非常小，则可以考虑增加堆大小。

选择不适合应用程序的堆实现也会造成麻烦。如“配置存储器”部分所述，有五种不同的堆实现。其中一些堆实现不会释放已分配的存储器，而其他堆实现在分配随机大小的存储器方面存在问题。必须考虑到特定应用程序的堆要求。有关每个堆实现的更多信息，请参见 www.freertos.org/a00111.html。

5.2 检查栈是否溢出

客户最常求助的一个问题是栈溢出，但 FreeRTOS 提供了一些功能来帮助调试此类问题。

当任务尝试将更多的元素推入堆栈而没有足够的空间时，就会发生栈溢出。请在创建任务时指定栈大小。要确定任务是否即将使栈溢出，可以使用 `uxTaskGetStackHighWaterMark(TaskHandle_t xTask)` 函数。该函数将返回未使用栈的最小值（以字为单位）。如果使用 8 位 MCU，则返回值为 1 表示 1 个字节。对于 32 位 MCU，1 个字表示 4 个字节。要使用此函数，必须在配置文件中将 `INCLUDE_uxTaskGetStackHighWaterMark` 置 1。

此外，也可以在运行时检查栈是否溢出。通过将 `configCHECK_FOR_STACK_OVERFLOW` 置 1，内核可检查栈指针是否仍在有效的栈空间内。如果不在，将调用栈溢出钩子函数。钩子函数是允许应用程序在发生某些事件时进行响应并提供不同行为的函数。例如，当栈溢出时，可以打开 LED。另外，也可以打印错误消息并重启系统。应用程序必须提供 `vApplicationStackOverflowHook()` 函数。

有关 FreeRTOS 和栈溢出的更多信息，请参见 www.freertos.org/Stacks-and-stack-overflow-checking.html。

5.3 跟踪

Tracealyzer（前称为 FreeRTOS+Trace）是一种分析工具，可收集关于嵌入式应用程序行为的数据。在对应用程序进行故障排除或优化其性能时，此数据非常有用。Tracealyzer 以图形化方式显示何时运行哪些任务，这有助于发现饥饿等问题。有关如何在 Atmel Studio 中使用 Tracealyzer 的说明，请参见应用笔记 FreeRTOS™ Using Percepio® Trace on ATmega4809。如需了解更多信息，请参见 www.freertos.org/FreeRTOS-Plus/FreeRTOS_Plus_Trace/RTOS_Trace_Instructions.shtml。

6. 演示

可通过以下链接从 Atmel | START 中获取 ATmega4809 的 FreeRTOS 示例：http://start.atmel.com/#examples/ATmega4809/FreeRTOS。所需的硬件包括 ATmega4809 Xplained Pro 以及通过 EXT3 连接的 OLED1 Xplained Pro。当在 OLED 显示屏上显示时间时，演示应用程序会使 LED 闪烁。按下按钮还将点亮相应的 LED，同时更新 OLED 显示以反映最新的按钮事件。

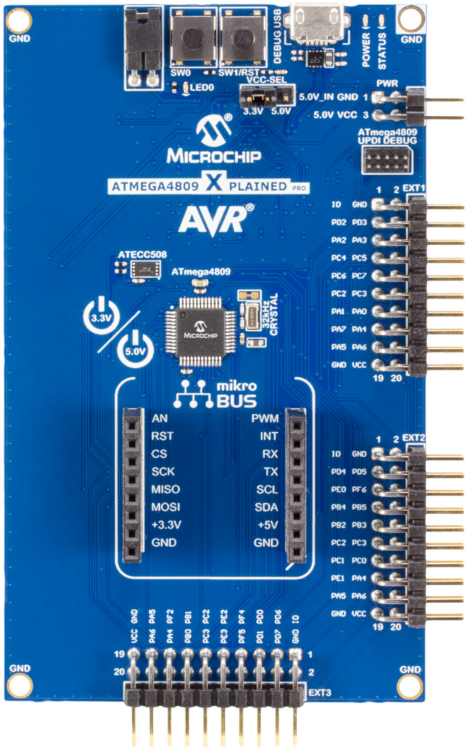
6.1 所需硬件

Atmel | START 示例项目所需的硬件如下所述。

6.1.1 ATmega4809 Xplained Pro 评估工具包

ATmega4809 Xplained Pro 评估工具包是一个用于评估 ATmega4809 AVR®单片机（MCU）的硬件平台。

图 6-1. ATmega4809 Xplained Pro

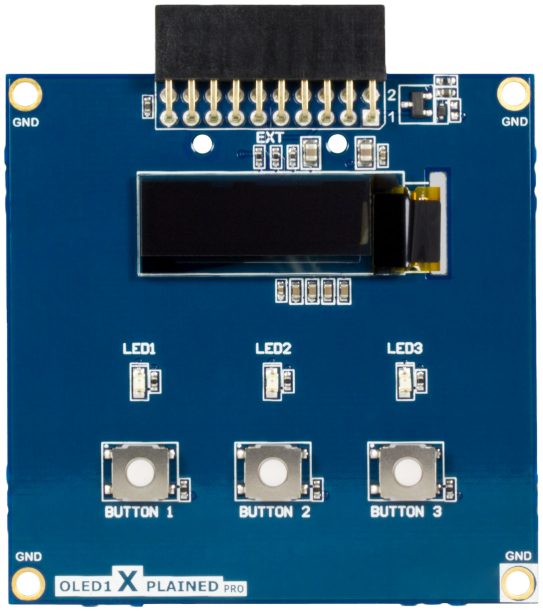


网页：www.microchip.com/developmenttools/ProductDetails/atmega4809-xpro。

6.1.2 OLED1 Xplained Pro 扩展工具包

OLED1 Xplained Pro 是一个扩展工具包，带有 128x32 分辨率的 OLED 显示屏、三个 LED 和三个按钮。它连接至任一 Xplained Pro 评估工具包的扩展插座。

图 6-2. OLED1 Xplained Pro



网页：www.microchip.com/developmenttools/ProductDetails/ATOLED1-XPRO。

6.2 划分为多个任务

示例应用程序的总体目标是使 LED 闪烁、处理按钮按下动作并在 OLED 屏幕上显示时间。请重温“从 RTOS 开发人员角度思考”章节中所述的思考方法。根据情况，不同的作业将被划分为多个单独的任务。

在考虑应用程序应执行的操作时，建议设置一个单独的任务来使 LED 每隔 250 ms 闪烁一次。另外，处理按钮按下动作也应是一个单独的任务。此外，还需要确定时钟的处理方式，并找到一种既能写入 OLED，又不会因为同时写入而引起冲突的方法。

示例使用了七个任务、两个队列、一个互斥、一个流缓冲区和一个报文缓冲区。有关如何实现这些任务的详细信息，请参见“实现”部分。下面简要说明了每项任务及其使用的通信方法。

状态 LED 任务

ATmega4809 Xplained Pro 上的 LED0 每 250 ms 闪烁一次。不使用通信。

键盘任务

检查其中一个按键（按钮）的状态是否改变，即按钮是已按下还是已释放。将更新的按键状态发送到名为 `key_queue` 的队列。

主任务

通过 `key_queue` 接收更新的按键状态，并创建要在 OLED 屏幕上打印的字符串。在写入之前，该任务会要求通过互斥保护名为 `oled_semaphore` 的屏幕。此外，该任务还会将按钮按下的信息发送到 `led_queue`。

LED 任务

通过 `led_queue` 接收有关最新按钮事件的信息，并相应地设置 LED 的状态。

时钟任务

每秒递增一次时间，并在获取 `oled_semaphore` 之后将时间写入到 OLED 屏幕。如果需要，也可以设置新的时间。

终端发送任务

通过名为 `terminal_tx_buffer` 的报文缓冲区接收信息，然后将该信息放入 USART TX 缓冲区中。

终端接收任务

通过名为 `terminal_rx_buffer` 的流缓冲区接收信息，并且可以根据接收到的信息请求时钟任务设置新的时间。

6.3 共享资源

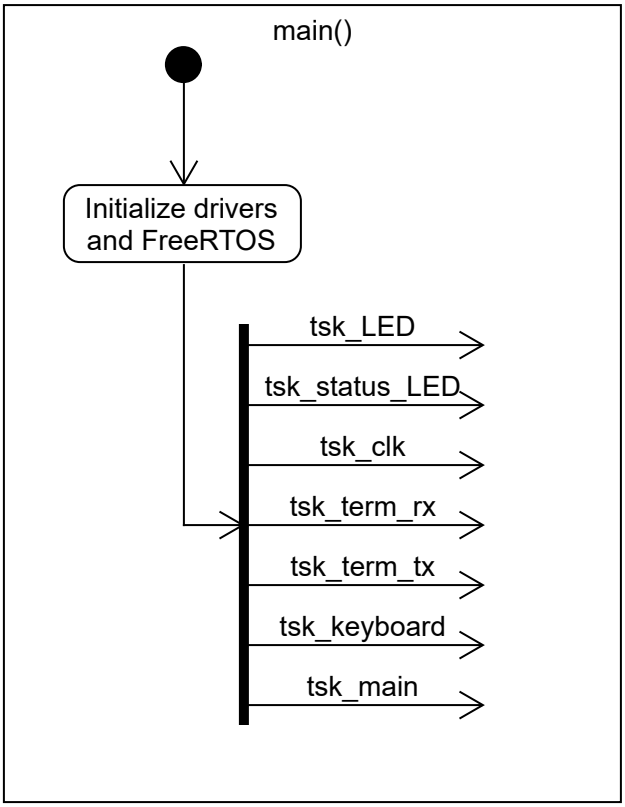
本应用程序中的共享资源是 OLED 屏幕。主任务和时钟任务都希望写入显示屏，但一次只能有一个任务执行此操作。如果两个任务试图同时写入屏幕，则程序可能会崩溃，或者可能出现一些不可预测的行为。因此，写入 OLED 的过程受互斥保护。

6.4 实现

本部分将简要说明各任务的实现。此外，还为每个任务提供了 UML 图。UML 的全称是 Unified Modeling Language（统一建模语言），可提供一种标准方式来可视化系统的设计。在本文档使用的图表中，与 FreeRTOS 相关的部分用灰色表示。

驱动程序和 FreeRTOS 的初始化以及任务的创建在主函数中完成，如下图所示。

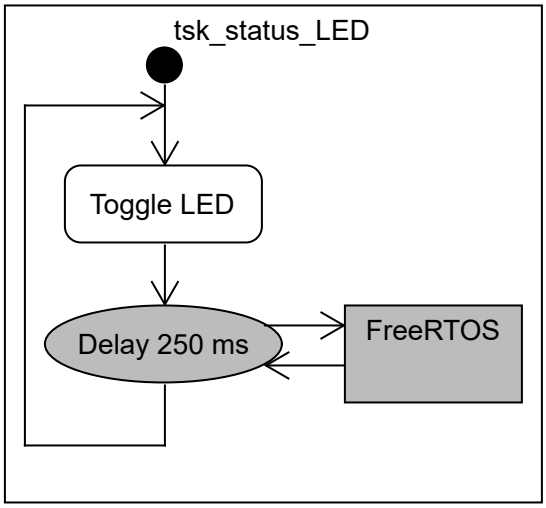
图 6-3. 任务和驱动程序的初始化



6.4.1 状态 LED 任务

这也许是应用程序中最简单的任务。它仅包含 LED 切换函数，延时为 250 ms。延时函数采用节拍数作为输入参数。要将 ms 转换为节拍数，可使用函数 `pdMS_TO_TICKS(250)`。UML 图表如下图所示。

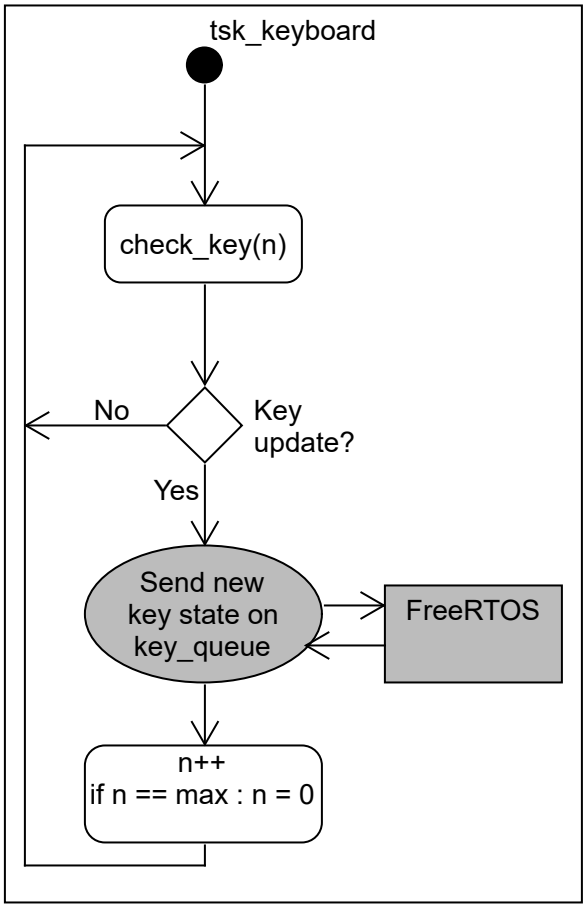
图 6-4. 状态 LED 任务



6.4.2 键盘任务

键盘任务 `tsk_keyboard()`用于处理按钮按下动作。如果其中一个按钮的状态更改，则将新状态放入 `key_queue`。`key_queue` 的接收者是下一部分中所述的主任务。UML 图表如下图所示。

图 6-5. 键盘任务

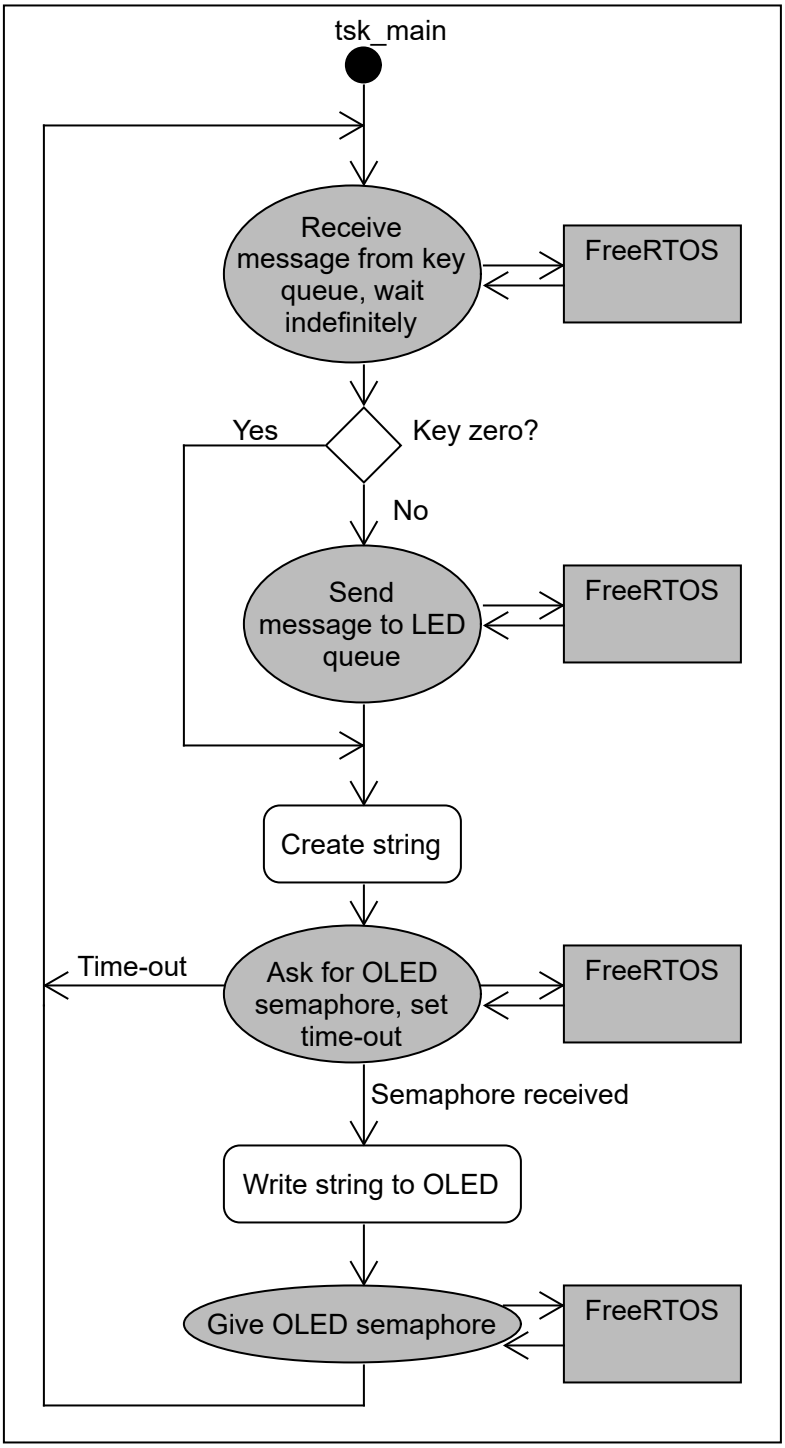


6.4.3 主任务

`tsk_main()`任务接收通过键盘任务放入 `key_queue` 的内容。根据接收到的信息，该任务将创建一个字符串来反映最新的按钮事件。然后，将该字符串写入 OLED。例如，字符串“Button 2 Released”和“Button 1 Pushed”。

此外，主任务还会将按钮按下的信息发送到 `led_queue`，以使相应的 LED 点亮。有关主任务的 UML 图表，请参见下图。

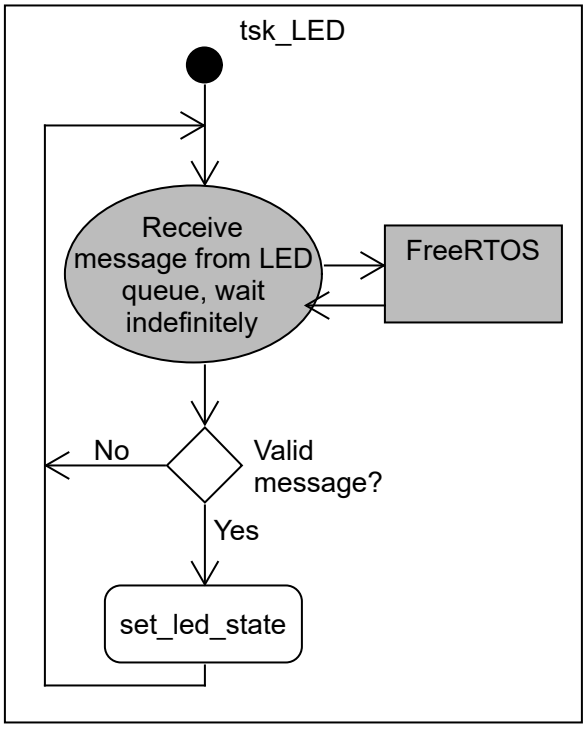
图 6-6. 主任务



6.4.4 LED 任务

LED 任务负责点亮按下的按钮对应的 LED，以及熄灭未按下按钮对应的 LED。只要按住按钮，LED 就会一直点亮。通过 `led_queue` 接收有关最新按钮事件的信息。如前文所述，主任务负责将信息发送至此队列。UML 图表如下图所示。

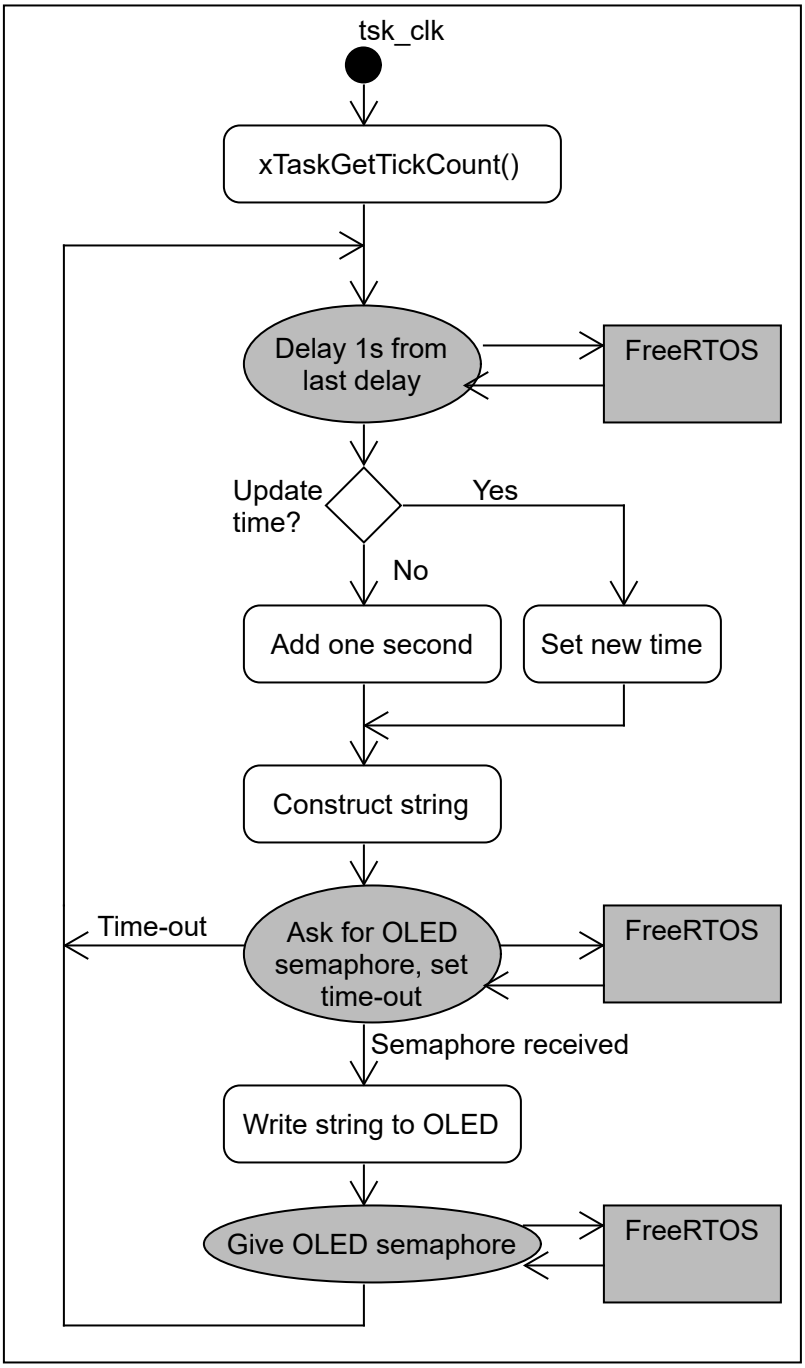
图 6-7. LED 任务



6.4.5 时钟任务

时钟任务负责每秒将时间写入 OLED 一次。在写入时，通过获取 `oled_semaphore` 完成此操作。此外，时钟任务还负责每秒将时间递增一次。UML 图表如下图所示。

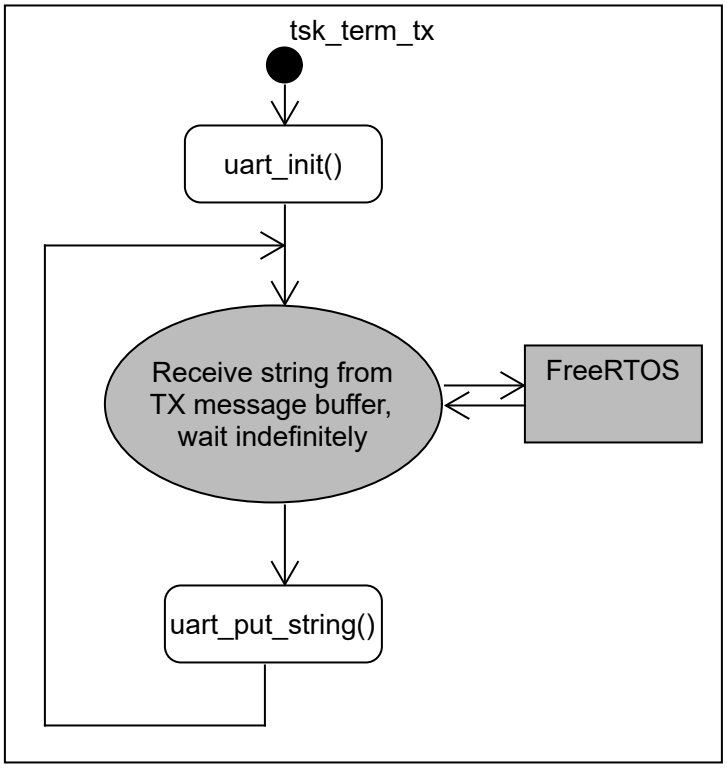
图 6-8. 时钟任务



6.4.6 终端发送任务

终端发送任务负责通过 USART 将其接收到的数据发送到称为 `terminal_tx_buffer` 的报文缓冲区，如下图所示。

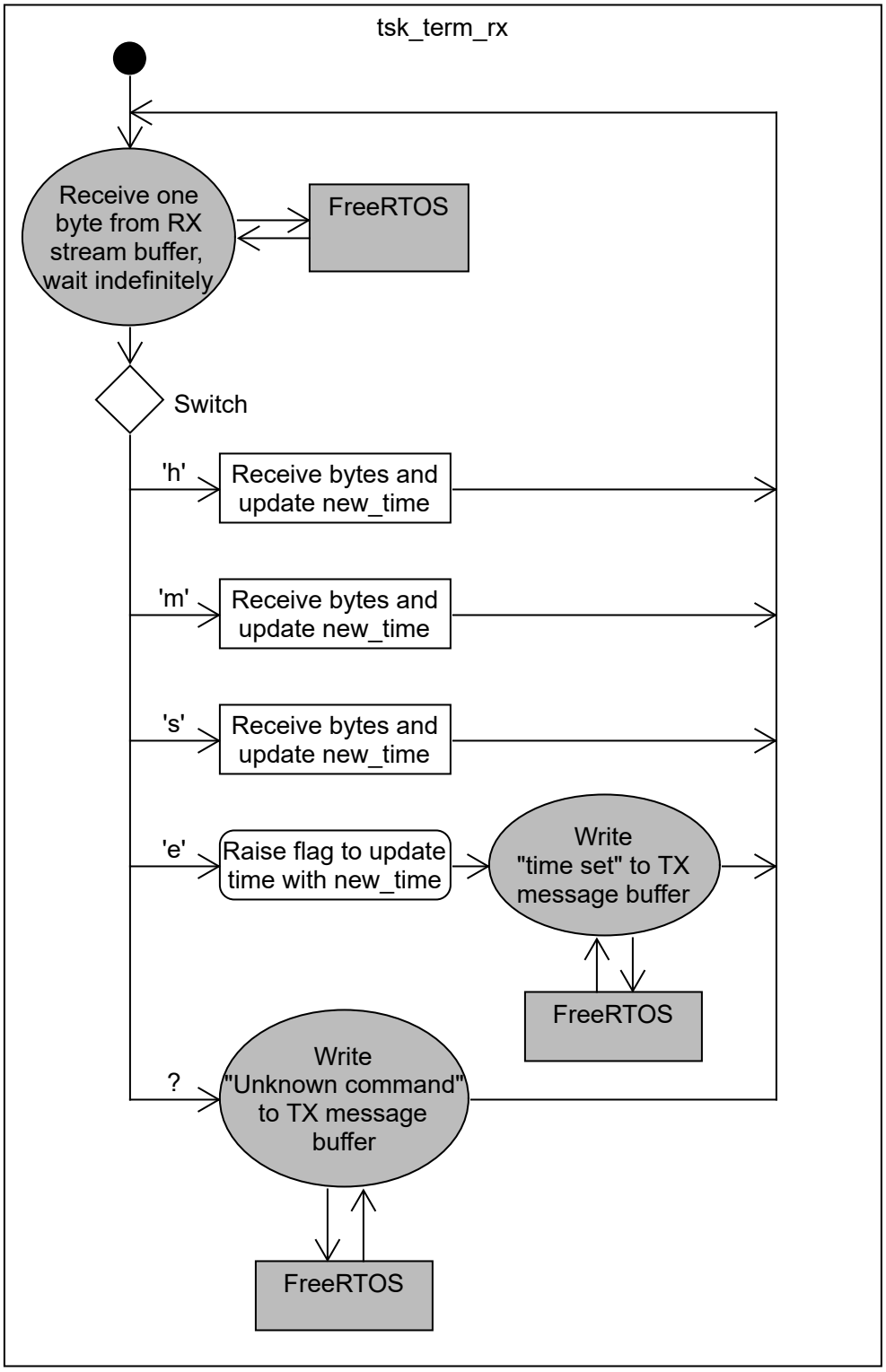
图 6-9. 终端发送任务



6.4.7 终端接收任务

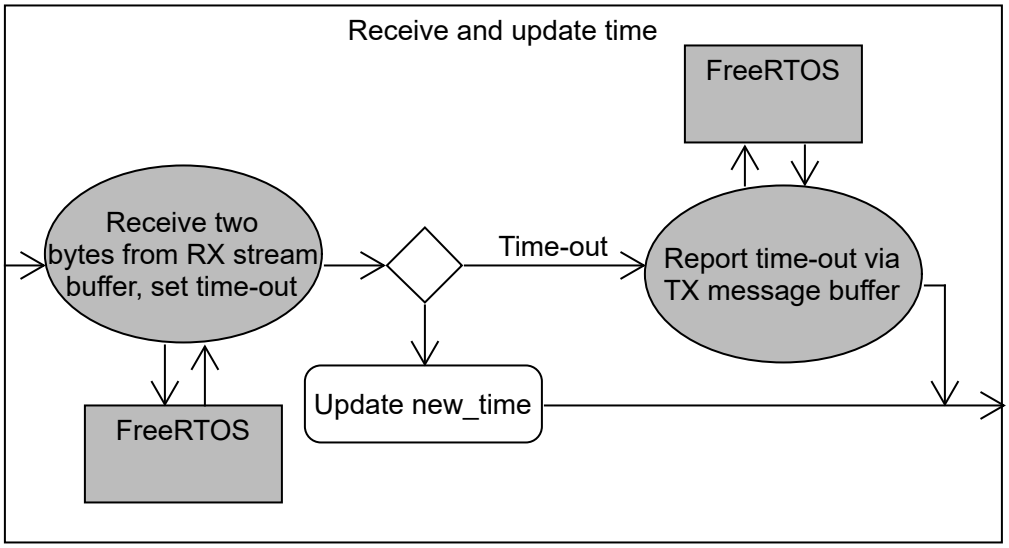
ISR 将接收到的 USART 数据放在称为 `terminal_rx_buffer` 的流缓冲区中。除非发生超时，否则终端接收任务将获取 `terminal_rx_buffer` 上接收到的数据，并根据接收到的消息调用更新时间函数。无论是设置了时间还是请求超时，终端接收任务都将通过 `terminal_tx_buffer` 将最新事件报告回终端发送任务。UML 图表如下图所示。

图 6-10. 终端接收任务



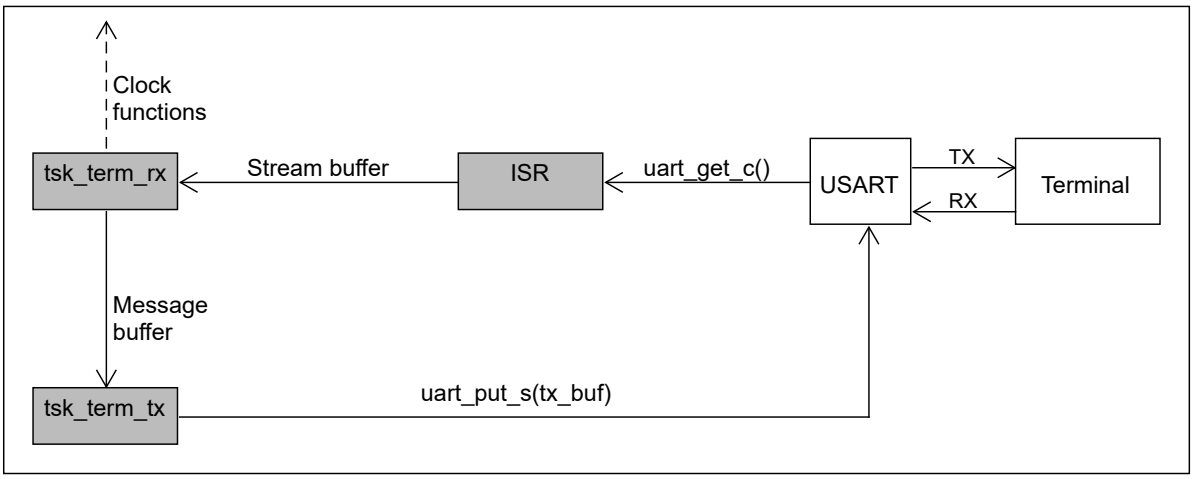
下图更详细地说明了“接收字节和更新 new_time”块。

图 6-11. 接收字节和更新时间



下图描述了两个终端任务之间的通信以及 USART 通信。

图 6-12. 终端发送任务和终端接收任务之间的通信



7. 从 Atmel | START 获取源代码

示例代码可通过 Atmel | START 获得，Atmel | START 是一种基于 Web 的工具，可通过图形用户界面（Graphical User Interface，GUI）配置应用程序代码。可以通过下面提供的直接示例代码链接或 Atmel | START 起始页上的 *Browse examples*（浏览示例）按钮，下载 Atmel Studio 和 IAR Embedded Workbench®对应的代码。

Atmel | START 网页： start.atmel.com/

示例代码

ATmega4809 FreeRTOS 示例
有关详细信息和示例项目的相关信息，请单击 Atmel | START 中的 *User guide*（用户指南）。*User guide* 按钮可以在该网页中找到，方法是在 Atmel | START 项目配置器中的仪表板视图中单击项目名称。

Atmel Studio

在 Atmel | START 的示例浏览器中单击 *Download selected example*（下载所选示例），下载 Atmel Studio 对应的代码并保存为`.atzip` 文件。要从 Atmel | START 下载文件，单击 *Export project*（导出项目），然后单击 *Download pack*（下载数据包）。

双击下载的`.atzip` 文件，将项目导入到 Atmel Studio 7.0。

IAR Embedded Workbench

有关如何在 IAR Embedded Workbench 中导入项目的信息，请打开 Atmel | START User Guide（Atmel | START 用户指南），选择 *Using Atmel Start Output in External Tools*（使用外部工具中的 Atmel Start 输出），然后选择 *IAR Embedded Workbench*。单击 Atmel | START 起始页右上角的 *Help*（帮助）或项目配置器中右上角的 *Help And Support*（帮助和支持），均可找到 Atmel | START 用户指南的链接。

8. 版本历史

文档版本	日期	备注
B	2019 年 12 月	从“相关器件”章节中删除了 tiny0 和 tiny1。更新了“从 RTOS 开发人员角度思考”章节中的流程图。
A	2018 年 04 月	文档初始版本。

Microchip 网站

Microchip 网站（www.microchip.com/）为客户提供在线支持。客户可通过该网站方便地获取文件和信息。我们的网站提供以下内容：

- **产品支持**——数据手册和勘误表、应用笔记和示例程序、设计资源、用户指南以及硬件支持文档、最新的软件版本以及归档软件
- **一般技术支持**——常见问题解答（FAQ）、技术支持请求、在线讨论组以及 Microchip 设计伙伴计划成员名单
- **Microchip 业务**——产品选型和订购指南、最新 Microchip 新闻稿、研讨会和活动安排表、Microchip 销售办事处、代理商以及工厂代表列表

产品变更通知服务

Microchip 的产品变更通知服务有助于客户了解 Microchip 产品的最新信息。注册客户可在他们感兴趣的某个产品系列或开发工具发生变更、更新、发布新版本或勘误表时，收到电子邮件通知。

欲注册，请访问 www.microchip.com/pcn，然后按照注册说明进行操作。

客户支持

Microchip 产品的用户可通过以下渠道获得帮助：

- 代理商或代表
- 当地销售办事处
- 应用工程师（ESE）
- 技术支持

客户应联系其代理商、代表或 ESE 寻求支持。当地销售办事处也可为客户提供帮助。本文档后附有销售办事处的联系方式。

也可通过 www.microchip.com/support 获得网上技术支持。

Microchip 器件代码保护功能

请注意以下有关 Microchip 器件代码保护功能的要点：

- Microchip 的产品均达到 Microchip 数据手册中所述的技术规范。
- Microchip 确信：在正常使用的情况下，Microchip 系列产品非常安全。
- 目前，仍存在着用恶意、甚至是非法的方法来试图破坏代码保护功能的行为。我们确信，所有这些行为都不是以 Microchip 数据手册中规定的操作规范来使用 Microchip 产品的。这种试图破坏代码保护功能的行为极可能侵犯 Microchip 的知识产权。
- Microchip 愿与那些注重代码完整性的客户合作。
- Microchip 或任何其他半导体厂商均无法保证其代码的安全性。代码保护并不意味着我们保证产品是“牢不可破”的。代码保护功能处于持续发展中。Microchip 承诺将不断改进产品的代码保护功能。任何试图破坏 Microchip 代码保护功能的行为均可视为违反了《数字器件千年版权法案（Digital Millennium Copyright Act）》。如果这种行为导致他人在未经授权的情况下，能访问您的软件或其他受版权保护的成果，您有权依据该法案提起诉讼，从而制止这种行为。

法律声明

提供本文档的中文版本仅为了便于理解。请勿忽视文档中包含的英文部分，因为其中提供了有关 Microchip 产品性能和使用情况的有用信息。Microchip Technology Inc.及其分公司和相关公司、各级主管与员工及事务代理机构对译文中可能存在的任何差错不承担任何责任。建议参考 Microchip Technology Inc.的英文原版文档。

商标

ISBN： 978-1-5224-7253-7

质量管理体系

有关 Microchip 的质量管理体系的信息，请访问 www.microchip.com/quality。

全球销售及服务网点

美洲	亚太地区	亚太地区	欧洲
公司总部 2355 West Chandler Blvd. Chandler, AZ 85224-6199 电话：480-792-7200 传真：480-792-7277 技术支持： www.microchip.com/support 网址： www.microchip.com **亚特兰大** 德卢斯，佐治亚州 电话：678-957-9614 传真：678-957-1455 **奥斯汀，德克萨斯州** 电话：512-257-3370 **波士顿** 韦斯特伯鲁，马萨诸塞州 电话：774-760-0087 传真：774-760-0088 **芝加哥** 艾塔斯卡，伊利诺伊州 电话：630-285-0071 传真：630-285-0075 **达拉斯** 阿迪森，德克萨斯州 电话：972-818-7423 传真：972-818-2924 **底特律** 诺维，密歇根州 电话：248-848-4000 **休斯顿，德克萨斯州** 电话：281-894-5983 **印第安纳波利斯** 诺布尔斯维尔，印第安纳州 电话：317-773-8323 传真：317-773-5453 电话：317-536-2380 **洛杉矶** 米慎维荷，加利福尼亚州 电话：949-462-9523 传真：949-462-9608 电话：951-273-7800 **罗利，北卡罗来纳州** 电话：919-844-7510 **纽约，纽约州** 电话：631-435-6000 **圣何塞，加利福尼亚州** 电话：408-735-9110 电话：408-436-4270 **加拿大 - 多伦多** 电话：905-695-1980 传真：905-695-2078	**澳大利亚 - 悉尼** 电话：61-2-9868-6733 **中国 - 北京** 电话：86-10-8569-7000 **中国 - 成都** 电话：86-28-8665-5511 **中国 - 重庆** 电话：86-23-8980-9588 **中国 - 东莞** 电话：86-769-8702-9880 **中国 - 广州** 电话：86-20-8755-8029 **中国 - 杭州** 电话：86-571-8792-8115 **中国 - 香港特别行政区** 电话：852-2943-5100 **中国 - 南京** 电话：86-25-8473-2460 **中国 - 青岛** 电话：86-532-8502-7355 **中国 - 上海** 电话：86-21-3326-8000 **中国 - 沈阳** 电话：86-24-2334-2829 **中国 - 深圳** 电话：86-755-8864-2200 **中国 - 苏州** 电话：86-186-6233-1526 **中国 - 武汉** 电话：86-27-5980-5300 **中国 - 西安** 电话：86-29-8833-7252 **中国 - 厦门** 电话：86-592-2388138 **中国 - 珠海** 电话：86-756-3210040	**印度 - 班加罗尔** 电话：91-80-3090-4444 **印度 - 新德里** 电话：91-11-4160-8631 **印度 - 浦那** 电话：91-20-4121-0141 **日本 - 大阪** 电话：81-6-6152-7160 **日本 - 东京** 电话：81-3-6880-3770 **韩国 - 大邱** 电话：82-53-744-4301 **韩国 - 首尔** 电话：82-2-554-7200 **马来西亚 - 吉隆坡** 电话：60-3-7651-7906 **马来西亚 - 槟榔屿** 电话：60-4-227-8870 **菲律宾 - 马尼拉** 电话：63-2-634-9065 **新加坡** 电话：65-6334-8870 **台湾地区 - 新竹** 电话：886-3-577-8366 **台湾地区 - 高雄** 电话：886-7-213-7830 **台湾地区 - 台北** 电话：886-2-2508-8600 **泰国 - 曼谷** 电话：66-2-694-1351 **越南 - 胡志明市** 电话：84-28-5448-2100	**奥地利 - 韦尔斯** 电话：43-7242-2244-39 传真：43-7242-2244-393 **丹麦 - 哥本哈根** 电话：45-4485-5910 传真：45-4485-2829 **芬兰 - 埃斯波** 电话：358-9-4520-820 **法国 - 巴黎** 电话：33-1-69-53-63-20 传真：33-1-69-30-90-79 **德国 - 加兴** 电话：49-8931-9700 **德国 - 哈恩** 电话：49-2129-3766400 **德国 - 海尔布隆** 电话：49-7131-72400 **德国 - 卡尔斯鲁厄** 电话：49-721-625370 **德国 - 慕尼黑** 电话：49-89-627-144-0 传真：49-89-627-144-44 **德国 - 罗森海姆** 电话：49-8031-354-560 **以色列 - 若那那市** 电话：972-9-744-7705 **意大利 - 米兰** 电话：39-0331-742611 传真：39-0331-466781 **意大利 - 帕多瓦** 电话：39-049-7625286 **荷兰 - 德卢内市** 电话：31-416-690399 传真：31-416-690340 **挪威 - 特隆赫姆** 电话：47-72884388 **波兰 - 华沙** 电话：48-22-3325737 **罗马尼亚 - 布加勒斯特** 电话：40-21-407-87-50 **西班牙 - 马德里** 电话：34-91-708-08-90 传真：34-91-708-08-91 **瑞典 - 哥德堡** 电话：46-31-704-60-40 **瑞典 - 斯德哥尔摩** 电话：46-8-5090-4654 **英国 - 沃金厄姆** 电话：44-118-921-5800 传真：44-118-921-5820